Comparing Machine Learning Approaches to Predict Movie Ratings

Luke M.

6/23/2020

Introduction

Summary. This project compares several methods for predicting movie ratings. The goal of this project was to identify a modeling approach or an ensemble of modeling approaches that predicted movie ratings with a root mean squared error (RMSE) < 0.86490. Four machine learning (ML) methods were compared: regularization, stepwise regression on k-modes cluster averages, k-nearest neighbors, and random forests. The regularization method performed the best. **The regularization method had a final *RMSE of 0.86434*, which achieved the project goal's of obtaining an RMSE < 0.86490.** The regularization method alone performed better than any other individual method or any ensemble method examined here. Computationally, regularization was the simplest ML method examined, which demonstrated the broader ML adage that increasing computational complexity does not necessarily improve model performance.

Motivation. Movies have become a pervasive part of modern culture. As the popularity of movies has grown, so has the number and diversity of movies available to viewers. Recommending movies to viewers based on their preferences has become a valuable tool for helping viewers find enjoyable cinema. In 2009, Netflix launched a competition for data scientists to create a model to improve Netflix's movie recommendation system, and this competition served as the inspiration for many educational projects to train aspiring data scientists on how to create and compare ML methods. This report was a capstone deliverable for the HarvardX Data Science certificate.

Data Cleaning and Setup

Data Pre-processing. The data in this project were drawn from a sample of the MovieLens dataset, which included variables for: user ID, movie title, a list of the movie's genres, the movie's numerical identifier, the year the movie premiered, the time that the rating was given, and the rating that the viewer gave the movie. Our goal was to accurately predict movie ratings.

The models were trained and tested on training and test sets created from the "edx" dataset. Then, the trained models were applied to a separate set of observations (the validation data) that were not used to train the model. The train and test partitions of the edx dataset contained 80% and 20% of the edx observations, respectively.

```
#create index identifying the observations selected for the training dataset
train_index <- createDataPartition(edx$title, times = 1, p=.8, list = F)

#use the index and its inverse to create training and test sets
train <- edx[train_index,]
test <- edx[-train_index,]
```

The following code was used to parse the movie title column into title and year, and also to clean the data to make it more computationally efficient and statistically useful.

```
#creates columns with the movie year and rating year
edx[["movie_year"]] <- as.numeric(str_sub(edx$title,-5,-2))
```

```
edx<-edx %>% mutate(rating_year = year(as_datetime(timestamp)))
validation[["movie_year"]] <- as.numeric(str_sub(validation$title,-5,-2))
validation <- validation %>% mutate(rating_year = year(as_datetime(timestamp)))

#remove all dashes in the genres column. this will help the code run quickly
edx$genres<-gsub("-", "", edx$genres)
validation$genres<-gsub("-", "", validation$genres)

#convert several of the columns to factors for computational efficiency
edx[["movieId"]]<-as.factor(edx[["movieId"]])
edx[["title"]]<-as.factor(edx[["title"]])
edx[["genres"]]<-as.factor(edx[["genres"]])

#repeat the process for the validation data
validation[["movieId"]]<-as.factor(validation[["movieId"]])
validation[["title"]]<-as.factor(validation[["title"]])
validation[["genres"]]<-as.factor(validation[["genres"]])

#change factor levels of the validation set to match the edx data factor levels
levels(validation$movieId)<-levels(edx$movieId)
levels(validation$title)<-levels(edx$title)
levels(validation$genres)<-levels(edx$genres)
```

The "genres" column in the original data contained a concatenated list of all the genres which apply to each movie. The following code was used to split this genres column into a series of boolean columns–one boolean column for each possible genre. This laid the ground work for performing clustering analysis of the genres, as described later in the report.

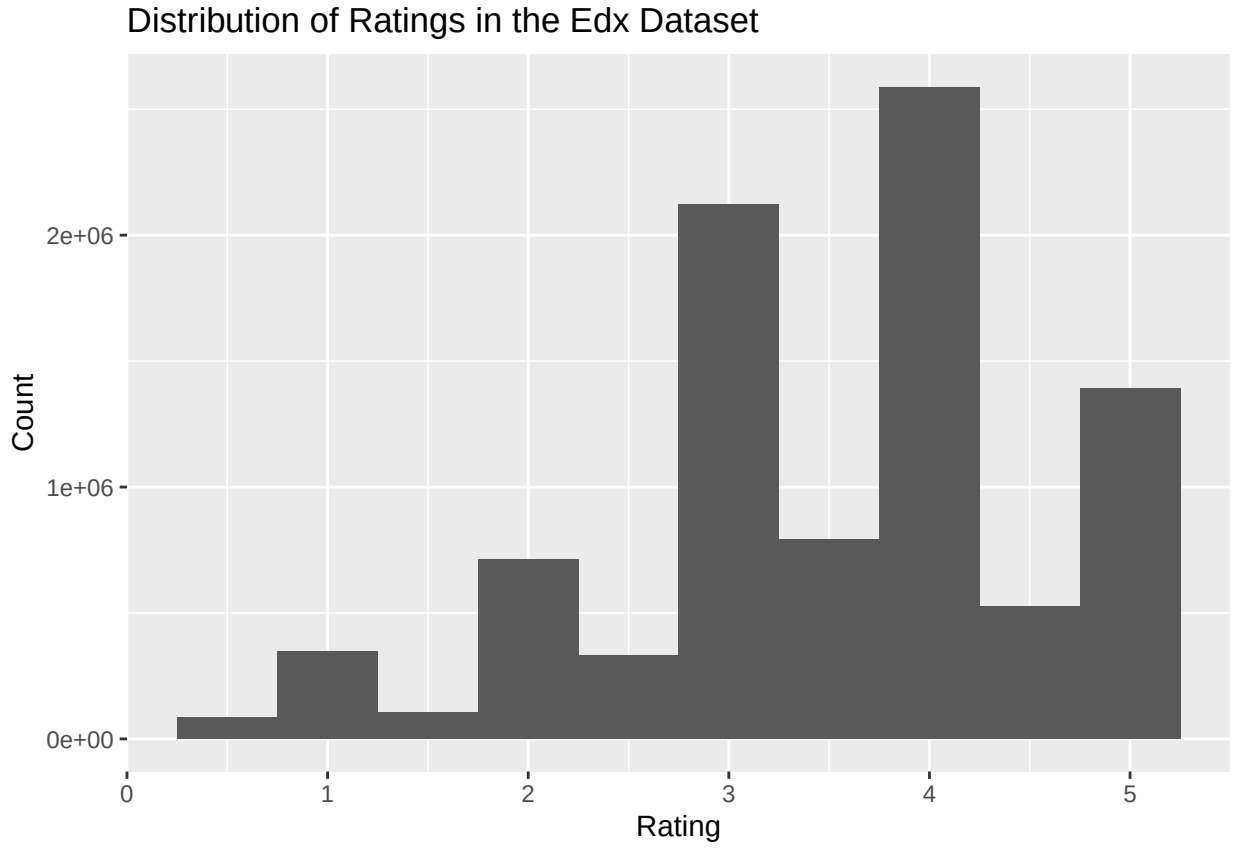
```
#create a list of all the entries in the data's original "genres" column
genre_levels<-levels(as.factor(edx$genres))

#split this list into a list of individual genres
splits<-sapply(genre_levels, function(g){
  strsplit(g,"[|]")[[1]]
})

#remove duplicate genres from the list created earlier
genres<-unique(unlist(splits, use.names=F))

#create a matrix listing each unique movie and the genres for each movie
titles_genres<-distinct(select(edx,c("title","genres")),title, .keep_all = T)
genres_matrix<-data.frame(unique(titles_genres$title))
names(genres_matrix)<-"title"
for (genre in genres){

  #append a new column that indicates if the movie in each row is in the genre
  genres_matrix[[genre]] <-str_detect(titles_genres$genres,genre)
}
```

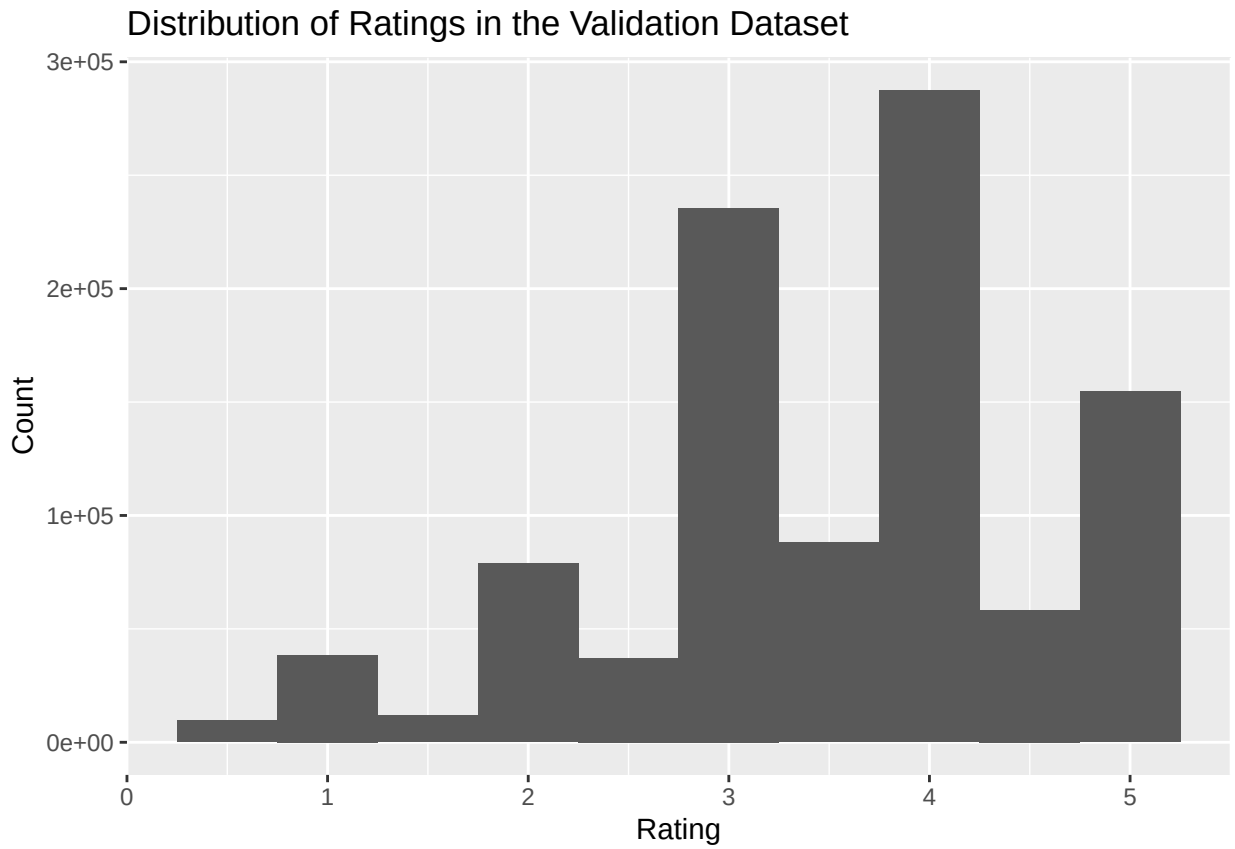
Computer Setup. The code for this project was written in R-4.0.1 and run on a Microsoft Azure virtual machine with 64GB RAM and 3.5 GHz processor. Most but not all of the R code for this project is printed in this report. The full R code is available in the accompanying .R file.

Exploratory Analysis

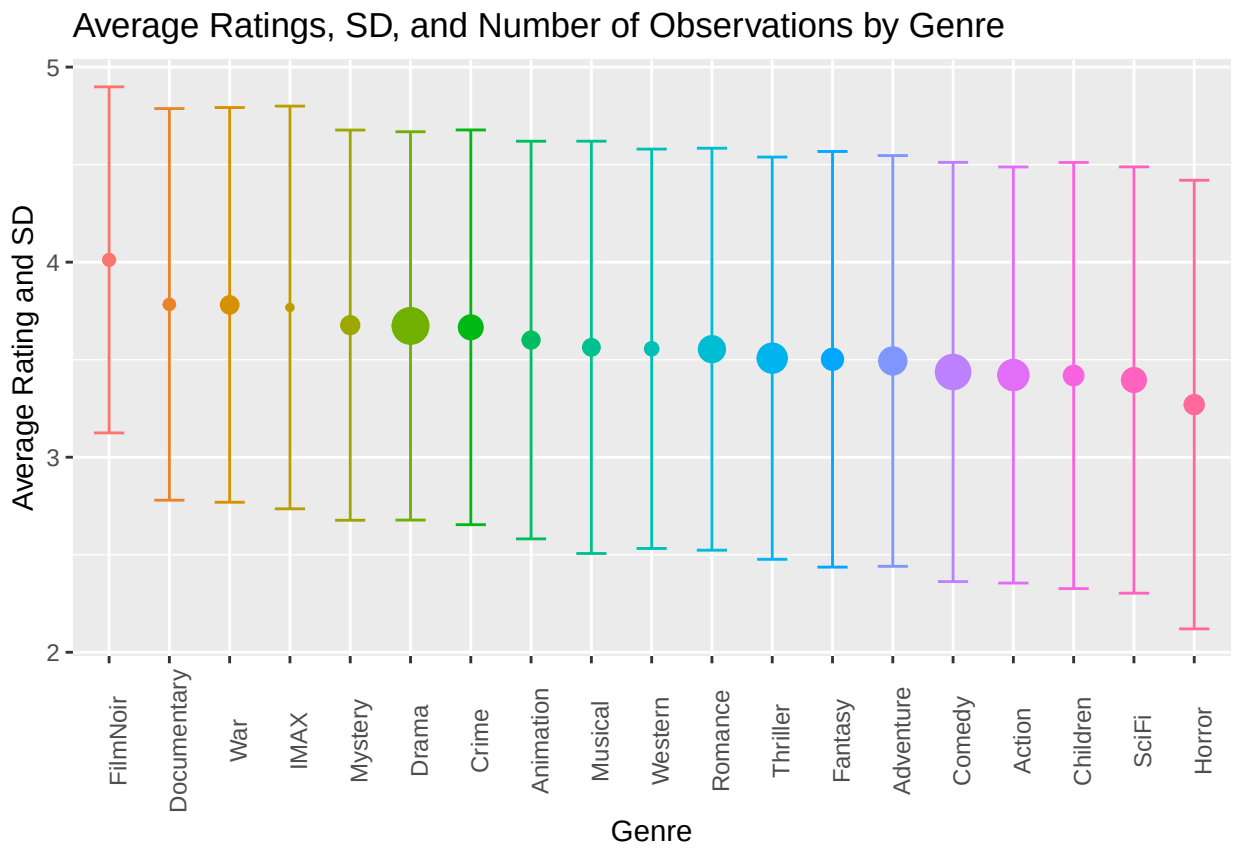
Descriptive Statistics for the Edx Dataset. The training and test datasets included a total of 10,676 distinct movies that premiered between 1915 and 2008, 69878 distinct users, and 9000055 distinct ratings given by users. The "genres" column in the data contained a list of genres applicable for each movie. Possible rating levels ranged from 0.5 stars to 5.0 stars in increments of 0.5 stars.

For the edx dataset which was used for training and testing, the average movie rating was 3.5124652 with SD=1.0603314. For the validation dataset, the average movie rating was 3.5120325 with SD=1.0612023. The distributions of ratings for the edx and validation datasets were compared using the graphs below. The graphs showed that the edx and validation datasets had very similar rating distributions.





Differences in Average Ratings For Each Genre. Based on the table below, it was clear that different genres tended to have different average movie ratings. The genres that received the highest ratings were Film-Noir, Documentary, War, Imax, and Mystery. The genres that received the lowest ratings were Horror, Sci-Fi, Children's films, Action, and Comedy. The average rating for the lowest-rated genre (Horror) was 0.742 starts below the average rating of the highest-rated genre (Film-Noir). This was almost one standard of deviation (SD = 1.06) difference between the highest and lowest rated genres, suggesting a movie's membership in certain genres might have affected the movie's average rating. This initial observation that a movie's membership in specific genres might affect its average rating helped motivate the design of the ML methods presented later in this report. However, the graph below showed that the standard deviations of ratings for each genre overlap, suggesting the differences in average ratings across genres might not be statistically significant, so further analysis was warranted. The red horizontal line in the graph below shows the global rating average for the training data. The size of the dots is proportional to the number of movies in each genre in the training data.



Correlations Between Ratings and Genres. The likelihood of a causal-link between a movie's membership in certain genres and that movie's average rating was further established by calculating the pairwise correlations between genres and ratings. Pearson's pairwise correlation coefficient is a common, easily interpreted method for estimating the extent to which two variables co-vary. The Pearson correlation coefficient can be thought of as the percent of two variables' variance that is correlated (i.e., the percent of variance that co-varies). Columns for genres with a low absolute correlation to movies' ratings were excluded from the ML models because these columns appeared at first blush to have little useful information on ratings, so including these genres have have merely increased the computational complexity of the model without improving its ability to accurately predict movie ratings.

Table 1: Correlation Coefficients for Genre Correlations to Ratings

genre	correlation_coeff	rank
Drama	0.1328069	1
War	0.0621016	2
Crime	0.0602064	3
FilmNoir	0.0543862	4
Mystery	0.0402868	5
Documentary	0.0261272	6
Animation	0.0194585	7
Romance	0.0189007	8
Musical	0.0107803	9
IMAX	0.0072605	10
Western	0.0060083	11
Thriller	-0.0026705	12
Fantasy	-0.0033589	13

genre	correlation_coeff	rank
Adventure	-0.0092628	14
Children	-0.0264247	15
SciFi	-0.0460698	16
Action	-0.0541538	17
Comedy	-0.0573893	18
Horror	-0.0660188	19

Methods

Baseline Models

Before applying various ML techniques, the performance of several simple models was estimated to establish a baseline of RMSEs to use to evaluate whether applying ML techniques meaningfully improved rating predictions. The following code was used to create tables to store and compare the RMSEs for each modeling method and to store the predictions from each method.

```
#create empty tables where rmse results for each model will be stored
train_rmse_table<-data.frame(model=character(0),rmse=double(0))
test_rmse_table<-data.frame(model=character(0),rmse=double(0))
validation_rmse_table<-data.frame(model=character(0),rmse=double(0))

#create dataframes where the y-hats from each model will be stored
train_results<-data.frame(title=train$title,rating=train$rating)
test_results<-data.frame(title=test$title,rating=test$rating)
validation_results<-select(validation,c(title,userId,rating))
```

Random Guesses. The first baseline, non-ML model examined was a model that randomly guesses movie ratings. This approach assumed all possible rating had the same probability of occurring. This random guessing method was expected to perform the worst and have the highest RMSE because this method did not base its predictions on any information gleaned from the training data.

```
#create list of possible ratings
poss_ratings<-seq(.5,5,.5)

#use random sample of possible ratings to generate predicted ratings and RMSEs
train_results[["y_hat_random_guesses"]]<-
  sample(poss_ratings, nrow(train),replace = T)

train_rmse_table[nrow(train_rmse_table)+1,]<-
  c("random_guesses",
    RMSE(train_results$y_hat_random_guesses,
         train_results$rating))

test_results[["y_hat_random_guesses"]]<-
  sample(poss_ratings, nrow(test),replace = T)

test_rmse_table[nrow(test_rmse_table)+1,]<-
  c("random_guesses",
    RMSE(test_results$y_hat_random_guesses,
         test_results$rating))

validation_results[["y_hat_random_guesses"]]<-
  sample(poss_ratings, nrow(validation), replace = T)
```

```
validation_rmse_table[nrow(validation_rmse_table)+1,]<-
  c("random_guesses",
    RMSE(validation_results$y_hat_random_guesses,
         validation_results$rating))
```

Weighted Guesses. This method also randomly assigns predicted ratings for each movie, but this method assumed each particular rating level had a specific probability of occurring, judging from the differential frequency of ratings observed in the training data. This method therefore estimated the probability that each rating should be selected for predictions based on that the proportion of movies that received that rating in the training data. In other words, the training data was used to estimated the prior probability of each rating level.

```
#estimate the proportion of observations in the training data that have each rating level
probs<-train %>% group_by(rating) %>%
  summarize(prob_per_rating=as.double(n()/nrow(train)))

#assign predictions at each rating level according their priors and find RMSEs
train_results[["y_hat_weighted_guesses"]]<-
  sample(poss_ratings, nrow(train), replace = T, prob=c(probs$prob_per_rating))

train_rmse_table[nrow(train_rmse_table)+1,]<-
  c("weighted_guesses",
    RMSE(train_results$y_hat_weighted_guesses,
         train_results$rating))

test_results[["y_hat_weighted_guesses"]]<-
  sample(poss_ratings, nrow(test), replace = T, prob=c(probs$prob_per_rating))

test_rmse_table[nrow(test_rmse_table)+1,]<-
  c("weighted_guesses",
    RMSE(test_results$y_hat_weighted_guesses,
         test_results$rating))

validation_results[["y_hat_weighted_guesses"]]<-
  sample(poss_ratings, nrow(validation), replace = T, prob=c(probs$prob_per_rating))

validation_rmse_table[nrow(validation_rmse_table)+1,]<-
  c("weighted_guesses",
     RMSE(validation_results$y_hat_weighted_guesses, validation_results$rating))
```

Global Average. The next baseline model used the overall average rating observed in the training data to predict all ratings in the test and validation data. The average rating of all observations in the training set was 3.512181, which in this baseline method was the predicted rating for all observations in the test and validation sets.

```
#make predictions solely from overall average rating in the training dataset
train_results[["y_hat_global_mean"]]<-mean(train$rating)
train_rmse_table<-rbind(train_rmse_table,c("global_mean",
                                           RMSE(train_results$y_hat_global_mean,
                                                train_results$rating)))

test_results[["y_hat_global_mean"]]<-mean(train$rating)
test_rmse_table<-rbind(test_rmse_table,c("global_mean",
                                         RMSE(test_results$y_hat_global_mean,
                                              test_results$rating)))
```

```
validation_results[["y_hat_global_mean"]]<-mean(train$rating)
validation_rmse_table<-rbind(validation_rmse_table,
                             c("global_mean",
                               RMSE(validation_results$y_hat_global_mean,
                                    validation_results$rating)))
```

Movie Average. This method calculated each movie's average rating in the training data, and then used each movie's estimated average rating to predict the movie's ratings in the test and validation datasets. This method assumed that the users in the training and testing sets gave the same average rating to the same movies.

```
#group the training dataset by movie and calculate average rating for each movie
train <- train %>% group_by(title) %>% mutate(movie_avg=mean(rating))
```

```
#predict ratings based on the movie's average rating in the training dataset
train_results[["y_hat_movie_mean"]]<-train$movie_avg
train_rmse_table<-rbind(train_rmse_table,c("movie_mean",
                                           RMSE(train_results$y_hat_movie_mean,
                                                train_results$rating)))

#create table listing each movie's average rating in the training dataset
avg_ratings<-train %>% ungroup() %>% select(c("movie_avg","title"))
avg_ratings<-distinct(avg_ratings)

#predict ratings based on each movie's average ratings in the training data
test <- left_join(test, avg_ratings, by="title")
test_results<-left_join(test_results, avg_ratings, by="title")

#rename the column to match naming convention used in the results tables
colnames(test_results)[which(colnames(test_results) == 'movie_avg')] <-
  'y_hat_movie_mean'
test_rmse_table<-rbind(test_rmse_table,c("movie_mean",
                                         RMSE(test_results$y_hat_movie_mean,
                                              test_results$rating)))

#repeat this prediction method for the validation set
validation <- left_join(validation, avg_ratings, by="title")
validation_results <- left_join(validation_results, avg_ratings, by="title")

#for any movies in the validation dataset that weren't in the training dataset,
#predict rating based on overall average rating from the training observations
validation$movie_avg[is.na(validation$movie_avg)] <-mean(train$rating)
validation_results$movie_avg[is.na(validation_results$movie_avg)] <-mean(train$rating)

#rename the column to match naming convention used in the results tables
names(validation_results)[which(names(validation_results) == 'movie_avg')] <-
  'y_hat_movie_mean'
validation_rmse_table<-rbind(validation_rmse_table,
                             c("movie_mean",
                               RMSE(validation_results$y_hat_movie_mean,
                                    validation_results$rating)))
```

Machine Learning Approaches

Four machine learning methods were created and optimized to determine which achieved the best RMSE.

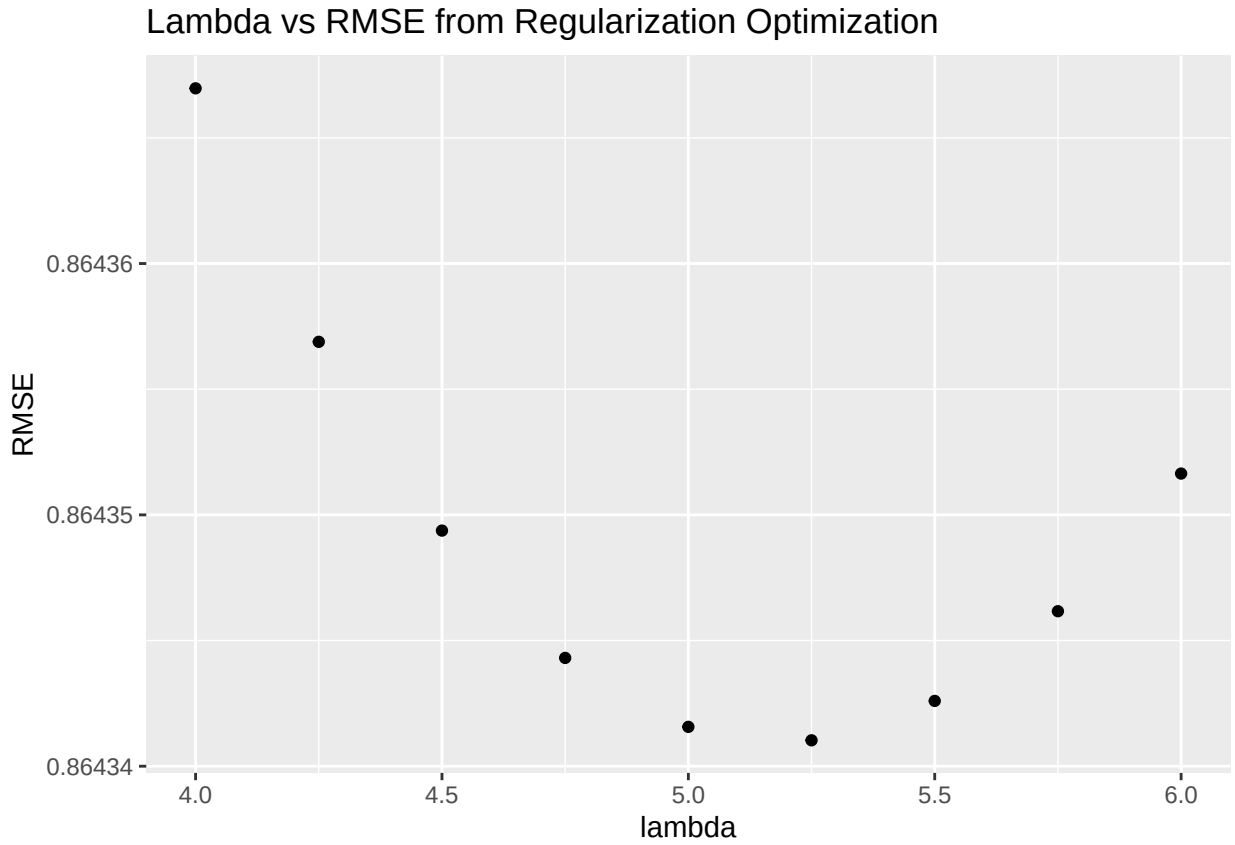
Regularization. The first ML method tested was regularization, which takes observation groups with small sample sizes and shifts their estimated ratings toward the overall rating mean. In this way, regularization helped reduce the effect of outliers, such as movies and genres with only a few ratings and such as users who only rated a few movies.

The regularization approach applied here regularized movie rating based on three main factors: number of ratings *by movie*, the number of ratings *by user*, and the number of ratings *by genre*. Predicted ratings for movies, users, and genres with relatively few observations were shifted toward the mean rating. A range of regularization parameters (lambda) was tested to determine the optimal extent to which the ratings should be shifted towards the overall mean.

```
#save the average rating from the training set as overall_mean
overall_mean<-mean(train$rating)

#create a list of lambdas to test
lambdas <- seq(4, 6, .25)

#use sapply to create a list of RMSEs achieved by each lambda
regularization_results<-sapply(lambdas,function(lambda){

  #create list of regularized movie effects
  reg_effects <<- train %>%
    group_by(title) %>%
    #all column for the overall mean rating
    mutate(overall_mean=overall_mean) %>%
    mutate(movie_effect = sum(rating - overall_mean)/(n()+lambda))

  #create list of regularized user effects
  reg_effects <<- reg_effects %>%
    group_by(userId) %>%
    mutate(user_effect = sum(rating - overall_mean - movie_effect)/(n()+lambda))

  #add list of regularized genre effects by looping through the list of genres
  reg_effects_copy <<- reg_effects
  genre_effects <<- data.frame(matrix(NA,ncol=1,nrow=nrow(train)))[-1]
  for (genre in genres_to_keep){
    #sum all of the genre effects that have been calculated so far
    reg_effects_copy$cumulative_genre_effects<-rowSums(genre_effects)

    #estimate the effect of the genre currently in the loop
    genre_eff <- reg_effects_copy %>%
      group_by_(genre) %>%
      mutate(genre_effect = sum(rating -
                                  overall_mean -
                                  movie_effect -
                                  user_effect -
                                  cumulative_genre_effects)/(n()+lambda)) %>%
      .$genre_effect

    #update the genre_effects matrix
    genre_effects<<-cbind.data.frame(genre_effects,genre_eff)
  }
```

```
  #create column that sums together all of the estimated genre effects
  reg_effects$cumulative_genre_effects<-rowSums(genre_effects)

  #create data frame with just the movie effects and genre effects for each movie
  movie_adjusments <- reg_effects %>% ungroup() %>%
    select(c(title, movie_effect, cumulative_genre_effects))
  movie_adjusments <- distinct(movie_adjusments)

  #create data frame with the user effects for each user
  user_adjustments <- reg_effects %>% ungroup() %>%
    select(c(userId, user_effect))
  user_adjustments <- distinct(user_adjustments)

  #bind the movie_adjustments and user_adjustments to test and validation sets
  test_copy <- test %>% left_join(movie_adjusments, by="title") %>%
    left_join(user_adjustments, by="userId")
  validation_copy <- validation %>% left_join(movie_adjusments, by="title") %>%
    left_join(user_adjustments, by="userId")

  #calculate y_hat for the test set
  y_hat_reg_train <- overall_mean + reg_effects$user_effect +
    reg_effects$movie_effect + reg_effects$cumulative_genre_effects

  y_hat_reg_test <- overall_mean + test_copy$user_effect +
    test_copy$movie_effect + test_copy$cumulative_genre_effects

  y_hat_reg_validation <- overall_mean + validation_copy$user_effect +
    validation_copy$movie_effect + validation_copy$cumulative_genre_effects

  list(lambda,RMSE(y_hat_reg_test,test$rating),
       y_hat_reg_train,
       y_hat_reg_test,
       y_hat_reg_validation)
})

#save the lambda that achieved the lowest RMSE
best_lambda<-unlist(
  regularization_results[1,which.min(regularization_results[2,])])
```

As visible in the graph below, the lowest RMSE occurred when lambda=5.25, which achieved an RMSE=0.864341.



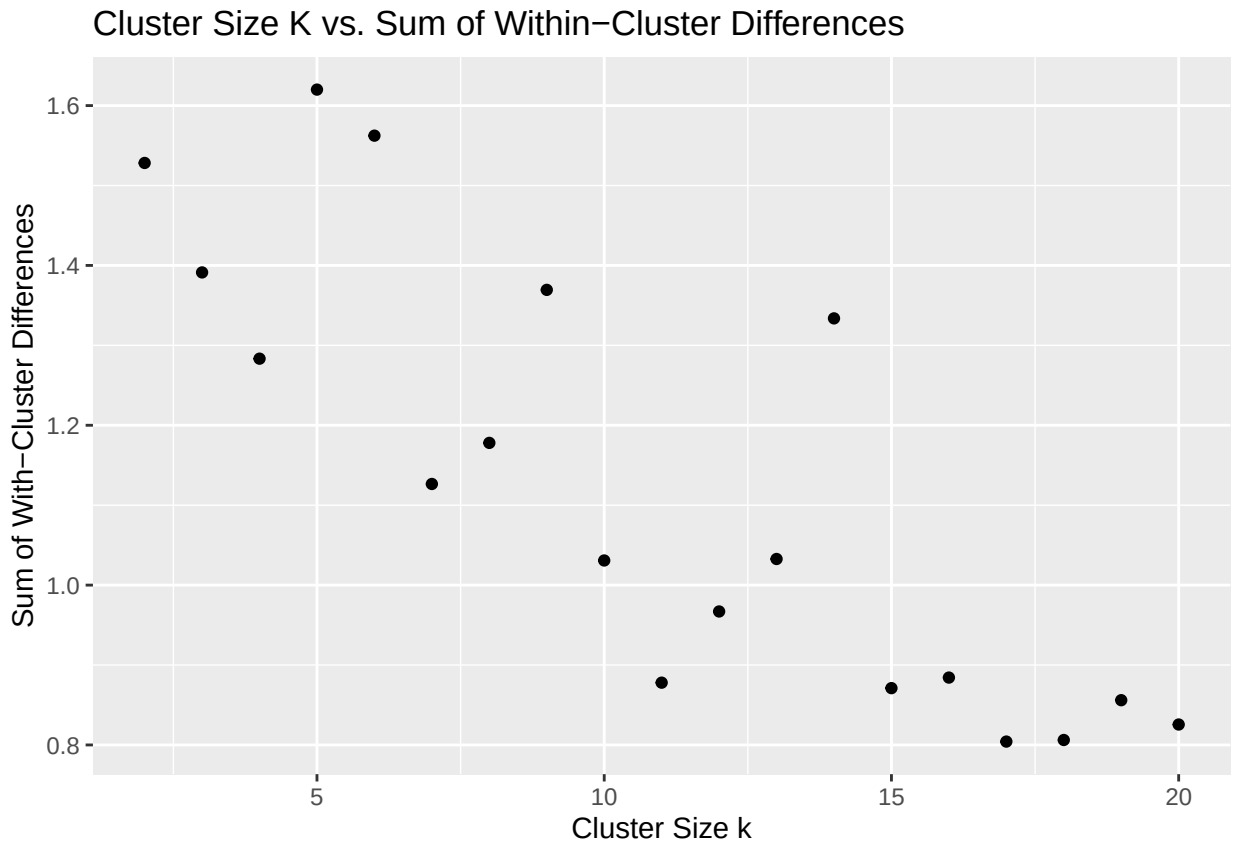

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   2429036  129.8    7106802  379.6  13880471  741.3
## Vcells 279620481 2133.4  719956562 5492.9 719638976 5490.5
```

K-Modes Clustering and Stepwise Regression. The winners of the 2009 Netflix Challenges used a clustering method to group movies into clusters of similar movies, which allowed the winning team to substantially improve their rating predictions. The exact clustering method they used remains unknown, but the code below represents an attempt to use clustering to improve predictions. A popular clustering method is known as k-means clustering, which sorts observations into clusters of observations with similar values for k-dimensional parameters of interest. K-means clustering is designed for k-dimensional continuous data, and a similar method known as k-modes was designed for categorical and boolean data. Here we applied the k-modes method because our genre membership columns for each genre were boolean.

In the code below, k specifies the number of clusters created. The optimal k is somewhat subjective, but it is generally considered to occur in the "elbow" in the graph of k vs *SSD* (the sum of squared differences among the observations in each cluster). This elbow occurs indicates the general value of k at which increasing the number of clusters would not significantly improve cohesion within clusters. In this context, cohesion is defined as the sum of within-cluster differences, which is a way of measuring overall similarity of observations in any given cluster. The theory is that clusters with highly similar observations generally will yield better cluster-based predictions.

```
set.seed(1)

#create list of possible values of k
k=seq(2,20,1)

#calculate the within-cluster sum of squares differences (ssd) for each k
ssd<-sapply(k,function(k_){
```

```
  kms<-kmodes(genres_matrix[2:ncol(genres_matrix)], k_, iter.max = 1,
              weighted = FALSE, fast = T)

    #calculate and return the sum of squared errors among rows of each cluster
  sum(kms$withindiff)/nrow(genres_matrix)
})

#this plot shows us that the "elbow" occurs somewhere between 10-15, so
ggplot(NULL, aes(x=k,y=ssd)) +
  geom_point() +
  theme_grey() +
  labs(x="Cluster Size k",
       y="Sum of With-Cluster Differences",
       title="Cluster Size K vs. Sum of Within-Cluster Differences")
```




```
#identify the best k that minimizes SSD for observations within clusters
best_k<-k[which(ssd==min(ssd))]

#using the optimal k to cluster the movies
clusters<-kmodes(genres_matrix[2:ncol(genres_matrix)], best_k,
                 iter.max = 5, weighted = FALSE, fast = T)

#print out the sizes of eachh cluster
clusters$size
```

```
## cluster
```

```
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
##  711  830  538 2721  446  292  282   79 3028  154   96  863   76   34  307  163
##   17
##   56
```

After each user's average cluster rating was appended to the datasets, a stepwise regression method was applied. This method is a ML technique that takes a list of possible parameters and builds a linear regression model that contains the combination of parameters that achieves the optimal model fit (i.e., achieves the lowest RMSE). The variables included in this project's stepwise regression method included the movie's average rating from all users, the average rating that each user gave to all movies they rated, user's cluster average rating, and the genre columns.

Because this stepwise regression method is computationally intensive, it was run on a subset of one fifth (20%) of the observations in the training and test datasets. The sample index was generated using the following code. R's train() method by default performs 25 bootstrap resamples.

```
#take sample of observations from the train set
sample_shrinkage_factor <- 5
sample_index<-sample(seq(1,nrow(train),1),
                     nrow(train)/sample_shrinkage_factor,
                     replace = F)
train_results_sample<-train_results[sample_index,]

#now take sample for test set
test_sample_index<-sample(seq(1,nrow(test),1),
                          nrow(test)/(sample_shrinkage_factor),
                          replace = F)
test_results_sample<-test_results[test_sample_index,]
```

The model was fit using the following code.

```
#estimate the best stepwise regression model based on 25 bootstraps
best_stepwise_fit<-train(rating ~ .,
                         method = "leapSeq",
                         data = train[sample_index,],
                         tuneGrid=data.frame(nvmax=seq(1,ncol(train)-1,1)))
```

```
## Subset selection object
## 15 Variables  (and intercept)
##                          Forced in Forced out
## ActionTRUE                   FALSE      FALSE
## ChildrenTRUE                 FALSE      FALSE
## ComedyTRUE                   FALSE      FALSE
## SciFiTRUE                    FALSE      FALSE
## DramaTRUE                    FALSE      FALSE
## HorrorTRUE                   FALSE      FALSE
## MysteryTRUE                  FALSE      FALSE
## CrimeTRUE                    FALSE      FALSE
## WarTRUE                      FALSE      FALSE
## DocumentaryTRUE              FALSE      FALSE
## FilmNoirTRUE                 FALSE      FALSE
## movie_avg                    FALSE      FALSE
## user_avg                     FALSE      FALSE
## genre_cluster                FALSE      FALSE
## genre_cluster_user_avg       FALSE      FALSE
## 1 subsets of each size up to 15
## Selection Algorithm: 'sequential replacement'
```

```
##           ActionTRUE ChildrenTRUE ComedyTRUE SciFiTRUE DramaTRUE HorrorTRUE
## 1  ( 1 )  " "        " "          " "        " "       " "        " "
## 2  ( 1 )  " "        " "          " "        " "       " "        " "
## 3  ( 1 )  " "        " "          " "        " "       "*"        " "
## 4  ( 1 )  " "        " "          "*"        " "       "*"        " "
## 5  ( 1 )  " "        " "          "*"        " "       "*"        " "
## 6  ( 1 )  " "        " "          "*"        " "       "*"        " "
## 7  ( 1 )  " "        " "          "*"        " "       "*"        " "
## 8  ( 1 )  "*"        " "          "*"        " "       "*"        "*"
## 9  ( 1 )  "*"        "*"          "*"        "*"       "*"        "*"
## 10  ( 1 ) "*"        " "          "*"        " "       "*"        "*"
## 11  ( 1 ) "*"        " "          "*"        "*"       "*"        "*"
## 12  ( 1 ) "*"        " "          "*"        "*"       "*"        "*"
## 13  ( 1 ) "*"        "*"          "*"        "*"       "*"        "*"
## 14  ( 1 ) "*"        "*"          "*"        "*"       "*"        "*"
## 15  ( 1 ) "*"        "*"          "*"        "*"       "*"        "*"
##           MysteryTRUE CrimeTRUE WarTRUE DocumentaryTRUE FilmNoirTRUE movie_avg
## 1  ( 1 )  " "          " "       " "     " "             " "          " "
## 2  ( 1 )  " "          " "       " "     " "             " "          "*"
## 3  ( 1 )  " "          " "       " "     " "             " "          "*"
## 4  ( 1 )  " "          " "       " "     " "             " "          "*"
## 5  ( 1 )  " "          " "       " "     " "             " "          "*"
## 6  ( 1 )  " "          "*"       " "     " "             " "          "*"
## 7  ( 1 )  " "          "*"       " "     "*"             " "          "*"
## 8  ( 1 )  " "          "*"       " "     " "             " "          "*"
## 9  ( 1 )  "*"          "*"       "*"     " "             " "          " "
## 10  ( 1 ) "*"          "*"       " "     "*"             " "          "*"
## 11  ( 1 ) "*"          "*"       " "     "*"             " "          "*"
## 12  ( 1 ) "*"          "*"       " "     "*"             " "          "*"
## 13  ( 1 ) "*"          "*"       "*"     "*"             "*"          "*"
## 14  ( 1 ) "*"          "*"       "*"     "*"             "*"          "*"
## 15  ( 1 ) "*"          "*"       "*"     "*"             "*"          "*"
##           user_avg genre_cluster genre_cluster_user_avg
## 1  ( 1 )  " "      " "           "*"
## 2  ( 1 )  " "      " "           "*"
## 3  ( 1 )  " "      " "           "*"
## 4  ( 1 )  " "      " "           "*"
## 5  ( 1 )  " "      "*"           "*"
## 6  ( 1 )  " "      "*"           "*"
## 7  ( 1 )  " "      "*"           "*"
## 8  ( 1 )  " "      "*"           "*"
## 9  ( 1 )  " "      " "           " "
## 10  ( 1 ) " "      "*"           "*"
## 11  ( 1 ) " "      "*"           "*"
## 12  ( 1 ) "*"      "*"           "*"
## 13  ( 1 ) "*"      " "           " "
## 14  ( 1 ) "*"      "*"           " "
## 15  ( 1 ) "*"      "*"           "*"

##    nvmax
## 15    15
```

KNN and Random Forest Methods. In addition to the aforementioned methods, k-nearest neighbor clustering (knn) and a random forest method were attempted but excluded from the final analysis because these two methods were too computationally intensive to execute quickly on a standard computer. Both

of these methods were attempted on a subset of about 5% of the observations in the edx training and test datasets. The code needed for these methods is included in the appendix.

The estimated RMSE for the random forest model was RMSE=0.88961, and the estimated RMSE for the knn model was RMSE=0.94775. The RMSE for an ensemble method combining these methods with regularization and stepwise regression was RMSE=0.86467 when estimated from the edx datasets. However, the RMSE was substantially higher (RMSE=1.16020) when the ensemble model was applied to the validation dataset. This disparity in the RMSE observed on the edx and validation datasets probably arose because the models were overtrained on the relatively small sample of 5% of observations taken from the edx dataset; the training data sample had too few observations to fit a model generalizable to the larger validation dataset.

Ensemble Model. An ensemble model was estimated by testing whether averaging the predictions of one or more of the aforementioned methods would reduce the overall RMSE. A key assumption underpinning ensemble modeling methods is that any given method will yield predictions with errors that *are normally distributed* around the true value of the predicted variable. Therefore, averaging the predictions of multiple models should in theory reduce overall RMSE by shifting the predictions toward the true values of the predicted parameter.

```
#rank the modeling methods according to their RMSEs
test_rmse_table[["rank"]]<-as.integer(rank(test_rmse_table$rmse))

#create a list of possible number of models
poss_num_models <- seq(2,nrow(test_rmse_table),1)

#estimate the RMSEs for ensemble models with various numbers of models included
rmses<- sapply(poss_num_models,function(poss_models){
  models_to_keep <- test_rmse_table %>% filter(rank<poss_models)
  results_to_keep <- lapply(list(models_to_keep$model), function(k){
    paste("y_hat",k,sep="_")
  })

  #average the models kept and calculate RMSE
  y_hat_meta_model_test <- rowMeans(select(test_results_sample,
                                           c(unlist(results_to_keep))))
  RMSE(y_hat_meta_model_test, test_results_sample$rating)
})
```

The one or more models listed in the table below were determined to be either the single model, when taken alone, or the *combination* of models that achieved the lowest RMSE.

```
## [1] 1
```

```
##            model              rmse rank
## 1 regularization 0.864341031104301    1
```

Results

The table below lists the final RMSEs for each individual modeling method, as well as the best ensemble method. Regularization performed the best, with an *RMSE of 0.86434*. The ensemble method in the table below lists a slightly lower RMSE, but this lower RMSE occurred because the ensemble method eliminated some rows from the dataset if they had any NA values. As show in the ensemble model analysis in the previous section, adding additional models to the regularization model did not improve overall model accuracy, so the best ensemble model only included predictions derived from the regularization model.

Of the four ML methods examined here, three (regularization, stepwise regression, and random forest) achieved RMSEs that were lower than the best baseline non-ML model (the movie mean model with RMSE=0.94258).

This demonstrated that using ML methods can help us to achieve better predictions that those generated by non-ML approaches.

model	rmse	rank
random_guesses	1.94059786052303	6
weighted_guesses	1.49979709847636	5
global_mean	1.05922337961736	4
movie_mean	0.942582491112189	3
regularization	0.864341031104301	1
stepwise_regression	0.902915998393181	2
meta_model	0.864021944988363	meta_model

Conclusion

This project achieved its overall goal by designing a ML model that predicted movie ratings with an *RMSE of 0.86434*, which met the goal of attaining a final RMSE < 0.86490. After comparing several non-ML and four ML methods, it was determined that the best method for predicting ratings was the regularization method, which predicted ratings based on the regularization effects by movie, user, and genres. This regularization method was computationally simple, and it achieved an RMSE=0.86434, which was superior to more complex ML methods and superior to any multi-method ensemble models that combined predictions from the various individual models examined here. Overall, this project demonstrates that simple ML approaches can help produce accurate, efficient recommendation systems, which have become an important part of modern life in general and cinema experiences in particular in the age of big data.

Appendix: Code for KNN and Random Forest Methods

Sampling Setup

```
#create a truncated df that has just columns required for the knn and rf methods
train <- train %>% ungroup() %>%
  select(-c("userId", "movie_year", "rating_year", "title","movieId","genres",
            "timestamp"))
test <- test %>% ungroup() %>%
  select(-c("userId","movie_year", "rating_year", "title","movieId","genres",
            "timestamp"))
validation <- validation %>% ungroup() %>%
  select(-c("userId", "movie_year","rating_year", "title","movieId","genres",
            "timestamp"))

#take a sample of observations from the train set
sample_shrinkage_factor <- 20
sample_index<-sample(seq(1,nrow(train),1),
                     nrow(train)/sample_shrinkage_factor, replace = F)
train_results_sample<-train_results[sample_index,]

#now take sample for test set
test_sample_index<-sample(seq(1,nrow(test),1),
                          nrow(test)/(sample_shrinkage_factor), replace = F)
test_results_sample<-test_results[test_sample_index,]

#clear computer's working memory
gc()
```

Random Forest Method

```
####random forest method
#create list of numbers of trees
trees<-seq(40,130,10)

#calculate fits for each number of trees
fits<-sapply(trees, function(tree){
  print(paste("Testing random forest model with",tree,"number of trees"))
  fit<-randomForest(as.matrix(train[2:ncol(train)])[sample_index,],
                    as.matrix(train$rating)[sample_index,], ntree=tree)
  list(tree,RMSE(fit$predicted,train_results_sample$rating),fit,fit$pred)
})

#plot the number of trees vs RMSE to determine the best number of trees
plot(fits[1,],fits[2,])
#TODO best_tree<-unlist(fits[1,which.min(fits[2,])])
print(paste("The random forest with the lowest RMSE has",
            unlist(fits[1,which.min(fits[2,])]),"trees with RMSE of",
            fits[2,which.min(fits[2,])]))

#add predicted values to results tables
best_rf_fit<-fits[3,which.min(fits[2,])]
train_results_sample[["y_hat_rf"]]<-unlist(predict(best_rf_fit,
                                                   train[sample_index,]))
test_results_sample[["y_hat_rf"]]<-unlist(predict(best_rf_fit,
                                                  test[test_sample_index,]))
validation_results[["y_hat_rf"]] <- unlist(predict(best_rf_fit,
                                                   validation))

#update rmse tables
train_rmse_table<-rbind(
  train_rmse_table,c("rf",
                     RMSE(train_results_sample[["y_hat_rf"]],
                          train_results_sample$rating)))
test_rmse_table<-rbind(
  test_rmse_table,c("rf",
                    RMSE(test_results_sample[["y_hat_rf"]],
                         test_results_sample$rating)))
validation_rmse_table<-rbind(
  validation_rmse_table,c("rf",
                          RMSE(validation_results[["y_hat_rf"]],
                               validation_results$rating)))
```

KNN Method

```
####knn method
#create list of possible Ks and possible ratings
ks=seq(10,100,10)
poss_ratings<-seq(.5,5,.5)

#estimate the model fit for each possible k
fits<-sapply(ks, function(k_){
  print(paste("Fitting the model with",k_,"number of observations per cluster"))
  fit<-knn3(rating ~ ., train[sample_index,], k=k_)
```

```
  print("Done fitting. Now using the model to make predictions.")
  y_hat<-predict(fit,test[test_sample_index,2:ncol(test)],type="prob")

  #for each row  find the rating or ratings with the highest probability
  #if more than one rating tied, take the mean of the ratings
  print("Calculating RMSE.")
  y_hats<-sapply(seq(1,nrow(y_hat),1),function(i){
    mean(poss_ratings[which(y_hat[i,]==max(y_hat[i,]))])
  })
  #clear the RAM to speed up R
  gc()
  #return k_ and RMSE
  list(k_,RMSE(y_hats,test_results_sample$rating),y_hats)
})

#plot the k vs the RMSE to determine the k that achieves the lowest RMSE
plot(fits[1,],fits[2,])
best_k_for_knn <-unlist(fits[1,which.min(fits[2,])])
print(paste("The k with the lowest RMSE is",unlist(
  fits[1,which.min(fits[2,])]),"with RMSE of",fits[2,which.min(fits[2,])]))

#store the fit object for the best knn fit
best_knn_fit <- knn3(rating ~ ., train[sample_index,], k=best_k_for_knn)

#append knn predictions to the test results
y_hat<-predict(best_knn_fit,train[sample_index,2:ncol(train)],type="prob")
train_results_sample[["y_hat_knn"]]<-sapply(seq(1,nrow(y_hat),1),function(i){
  mean(poss_ratings[which(y_hat[i,]==max(y_hat[i,]))])
})

#add results to test results table
test_results_sample[["y_hat_knn"]]<-unlist(fits[3,which.min(fits[2,])])

#add results to the validation results table
y_hat<-predict(best_knn_fit,validation[,2:ncol(train)],type="prob")
validation_results[["y_hat_knn"]]<-sapply(seq(1,nrow(y_hat),1),function(i){
  mean(poss_ratings[which(y_hat[i,]==max(y_hat[i,]))])
})

#calculate RMSEs and append them to the RMSE tables
train_rmse_table<-rbind(train_rmse_table,
                        c("knn",
                          RMSE(train_results_sample$y_hat_knn,
                               train_results_sample$rating)))
test_rmse_table<-rbind(test_rmse_table,
                       c("knn",fits[2,which.min(fits[2,])]))
validation_rmse_table<-rbind(validation_rmse_table,
                             c("knn",
                               RMSE(validation_results$y_hat_knn,
                                    validation_results$rating)))
```